From *Control of the Physical World by Intelligent Systems, Papers from the 1994 AAAI Fall Symposium,* B. Kuipers, ed., New Orleans, LA, November, 1994.

A Logical Approach to High-Level Robot Programming — A Progress Report*

Yves Lespérance, Hector J. Levesque, Fangzhen Lin, Daniel Marcu, Raymond Reiter, and **Richard B. Scherl**
Department of Computer Science
University of Toronto
Toronto, ON, M5S 1A4 Canada
{lesperan,hector,fl,marcu,reiter,scherl}@ai.toronto.edu

Abstract

This paper describes a novel approach to high-level robot programming based on a highly developed logical theory of action. The user provides a specification of the robot's basic actions (their preconditions and effects on the environment) as well as of relevant aspects of the environment, in an extended version of the situation calculus. He can then specify robot behaviors in terms of these actions in a programming language that allows references to world conditions (e.g. **if** $\exists c(\mathrm{POP\_CAN}(c) \wedge \mathrm{ON\_TABLE}(c))$ **then** $\mathrm{PICK\_UP}(c)$). The programs can be executed to drive the robot. The interpreter automatically maintains the world model required to execute programs based on the specification. The theoretical framework includes a solution to the frame problem and is very general — it handles dynamic and incompletely known worlds, as well as perception actions. Given this kind of domain specification, it is also possible to support more sophisticated reasoning, such as task planning at run-time. The specification can also be used to prove the robot control programs correct. A simple mail delivery application is used to present the approach. Ongoing work on implementing the approach and handling outstanding problems, such as modeling noisy sensors, is also described.

Introduction and Motivation

Virtually all current research in robotics concerns basic-level tasks like sensory processing, path planning, manipulator design and control, reactive agents, artificial insects, etc. In contrast, our ongoing research project in *cognitive robotics* is concerned with the theory and implementation of agents that reason, act and perceive in changing, incompletely known, unpredictable environments. Such agents must have higher level cognitive functions that involve reasoning, for example, about goals, actions, when to perceive and what to look for, the cognitive states of other agents, collaborative task execution, etc. In short, our concern is with integrating reasoning, perception and action within a uniform theoretical and implementation framework.

Our objective in this paper is to provide something of a progress report for this enterprise, and to demonstrate that, far from being the dead horse that some robotics researchers claim for logic and symbolic AI, ours is an extremely fruitful approach to the design and implementation of autonomous agents. The trick, on which earlier attempts at using logic for controlling robots foundered, is to get the technical foundations right. This paper outlines what we take to be appropriate foundations for a theory of action, and what consequences this has for robotics.

Logic and the Situation Calculus

The situation calculus is enjoying a new respectability these days, partly because its expressiveness is much richer than had been commonly believed (Gelfond, Lifschitz, & Rabinov 1991; Pinto 1994), partly because it has been found very useful for precisely characterizing the strengths and limitations of various general theories of actions (Lifschitz 1987). For both these reasons, we have grounded all our theorizing about cognitive robotics within the situation calculus. As we shall show in the remainder of this paper, this formalism admits very natural extensions to accommodate knowledge and perception, descriptions of complex behaviors, procedural control, etc. Moreover, it does so without sacrificing any of the clarity and logical precision which makes it the language of choice for theoretical analysis.

The Frame Problem

Despite much wishful thinking within the AI community, the frame problem will not go away. Our approach to cognitive robotics relies on a recent solution to this problem, one with particularly attractive theoretical and computational properties.

*This research received financial support from the Information Technology Research Center (Ontario, Canada), the Institute for Robotics and Intelligent Systems (Canada), and the Natural Science and Engineering Research Council (Canada). Levesque and Reiter are fellows of the Canadian Institute for Advanced Research.

Perception and Knowledge

Perceptual actions – see whether the door is open – are distinguished from ordinary ones – pick up the block – in that they influence an agent's state of knowledge about the world. This means that the right story about perception must provide an account of an agent's *epistemic state*, and how that is affected by perceptual actions. One of the nicer features of the situation calculus is that it leads quite naturally to an appropriate formalization of agent knowledge. Not surprisingly, perception contributes its own peculiarities to the frame problem. Later in this paper, we describe a suitable generalization of the earlier solution for ordinary actions to accommodate perception.

Complex Actions and Robot Control

Our results on the frame problem and the associated formal accounts of ordinary actions, knowledge and perceptual actions, apply only to *primitive, deterministic* actions. But the behavioral repertoire of a robot must include *complex* actions, for example the action of clearing off a table, defined as something like "While there is an object on the table, pick it up, then place it on the floor". This action is defined in terms of the primitive actions "pick up an object" and "put an object on the floor", using iteration and sequence. As we shall see, it is possible to define complex actions like this within the situation calculus. The result will be GOLOG, a novel logic programming language suitable for declaratively defining complex behaviors, but also capable of "executing" such actions. GOLOG is our candidate programming language for high-level robot control.

Integrating Reasoning, Action and Perception

By putting together the above results on the frame problem, perception and knowledge and complex actions, we obtain a unified theory combining these elements. We are now well positioned to investigate relationships that hold among reasoning, action and perception. These include the following kinds of tasks: (i) Planning to acquire knowledge, e.g. finding out Mary's telephone number. (ii) Introspection vs. perception for acquiring information. (iii) Goal driven perception: when to perceive, and what to look for.

Cognitive Robotics: How Are We Different?

Our approach to robotics is unabashedly neat.[1] We believe that much of the brittleness of current AI systems derives from a failure to provide a principled, theoretical account of the task to be achieved. Accordingly, we have adopted a firm methodological commitment in our project: No implementation without a situation calculus specification. We have surprised even ourselves with how far we have been able to push this idea. We now have specifications for various planning tasks and their associated planning algorithms, theories for sensor noise and uncertainty, indexicals, agent ability, STRIPS-like systems, time, etc. GOLOG, our robot programming language, has a situation calculus semantics. We have a functioning GOLOG interpreter, and implementations of a few simple applications (for the moment, in simulation mode only). Future plans include trying these ideas out to control a mobile robot.

[1] In this, we follow Kaelbling and Rosenschein (1990); but they focus on using logic to synthesize reactive agents, while we look at reasoning at run-time within user-specified control programs.

Outline of the Paper

A running example will be used throughout to illustrate our approach. It involves a mobile robot that is to deliver letters and small parcels within an office environment. We assume that the robot knows the general layout of the environment and the locations of peoples' offices. It receives requests to pick up items for delivery, say via electronic mail.

The next section covers the logical foundations of our approach. Then, we describe the GOLOG programming language. After that, we present our mail delivery example in detail. Then, we discuss issues of robot architecture and describe our experience so far in implementing and experimenting with GOLOG. We conclude by discussing topics for future research.

Logical Foundations

The Situation Calculus and the Frame Problem

The situation calculus (following the presentation in (Reiter 1991)) is a first-order language for representing dynamically changing worlds in which all of the changes are the result of named *actions* performed by some agent. Terms are used to represent states of the world–i.e. *situations*. If α is an action and s a situation, the result of performing α in s is represented by $do(\alpha, s)$. The constant S_0 is used to denote the initial situation. Relations whose truth values vary from situation to situation, called *fluents*, are denoted by predicate symbols taking a situation term as the last argument. For example, $\textsc{Robot\_carrying}(p, s)$ means that the robot is carrying package p in situation s. Functions whose denotations vary from situation to situation are called *functional fluents*. They are denoted by function symbols with an extra argument taking a situation term, as in $\textsc{pos}(\textsc{robot}, s)$, i.e., the robot's position in situation s.

It is assumed that the axiomatizer has provided for each action $\alpha(\vec{x})$, an *action precondition axiom* of the form given in 1, where $\pi_\alpha(\vec{x}, s)$ is a formula specifying the preconditions for action $\alpha(\vec{x})$.

Action Precondition Axiom

$$Poss(\alpha(\vec{x}), s) \equiv \pi_\alpha(\vec{x}, s) \qquad (1)$$

An action precondition axiom for the action DROP_OFF is given below.

$$\begin{aligned} &Poss(\text{DROP\_OFF}(pkg), s) \equiv \\ &\quad \text{ROBOT\_CARRYING}(pkg, s) \wedge \\ &\text{POS}(\text{ROBOT}, s) = \\ &\qquad \text{POS}(\text{IN\_BOX}(\text{RECIPIENT}(pkg)), s) \end{aligned} \tag{2}$$

This says that DROP_OFF is possible whenever the robot is carrying the package and is positioned at the recipient's "in" mailbox.

Furthermore, it is assumed that the axiomatizer has provided for each fluent F, two *general effect axioms* of the form given in 3 and 4.

General Positive Effect Axiom for Fluent F

$$Poss(a, s) \wedge \gamma_F^+(\vec{x}, a, s) \rightarrow F(\vec{x}, do(a, s)) \tag{3}$$

General Negative Effect Axiom for Fluent F

$$Poss(a, s) \wedge \gamma_F^-(\vec{x}, a, s) \rightarrow \neg F(\vec{x}, do(a, s)) \tag{4}$$

Here $\gamma_F^+(\vec{x}, a, s)$ is a formula describing under what conditions doing the action a in situation s leads the fluent $F(\vec{x})$ to become true in the successor situation $do(a, s)$ and similarly $\gamma_F^-(\vec{x}, a, s)$ is a formula describing the conditions under which performing action a in situation s results in the fluent $F(\vec{x})$ becoming false in situation $do(a, s)$.

For example, 5 is a positive effect axiom for the fluent ROBOT_CARRYING.

$$\begin{aligned} &Poss(a, s) \wedge a = \text{PICK\_UP}(package) \\ &\rightarrow \text{ROBOT\_CARRYING}(package, do(a, s)) \end{aligned} \tag{5}$$

Sentence 6 is a negative effect axiom for ROBOT_CARRYING.

$$\begin{aligned} &Poss(a, s) \wedge a = \text{DROP\_OFF}(package) \\ &\rightarrow \neg\text{ROBOT\_CARRYING}(package, do(a, s)) \end{aligned} \tag{6}$$

Effect axioms provide the "causal laws" for the domain of application.

Usually, the axiomatizer must also provide *frame axioms* that specify when fluents remain unchanged. The frame problem arises because the number of these frame axioms in the general case is of the order of $2 \times \mathcal{A} \times \mathcal{F}$, where $\mathcal{A}$ is the number of actions and $\mathcal{F}$ the number of fluents.

The solution of the frame problem (Reiter 1991; Pednault 1989; Schubert 1990; Haas 1987) rests on a *completeness assumption*. This assumption is that axioms 3 and 4 characterize all the conditions under which action a can lead to a fluent $F(\vec{x})$'s becoming true (respectively, false) in the successor situation. Therefore, if action a is possible and $F(\vec{x})$'s truth value changes from *false* to *true* as a result of doing a, then $\gamma_F^+(\vec{x}, a, s)$ must be *true* and similarly for a change from *true* to *false*. Additionally, *unique name axioms* are added for actions and situations.

Reiter (1991) shows how to derive a set of *successor state axioms* of the form given in 7 from the axioms (positive and negative effect) and the completeness assumption.

Successor State Axiom

$$\begin{aligned} &Poss(a, s) \rightarrow [F(\vec{x}, do(a, s)) \equiv \\ &\quad \gamma_F^+(\vec{x}, a, s) \vee (F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, s))] \end{aligned} \tag{7}$$

Similar successor state axioms may be written for functional fluents. A successor state axiom is needed for each fluent F, and an action precondition axiom is needed for each action a. Therefore only $\mathcal{F} + \mathcal{A}$ axioms are needed.

From 5 and 6 the following successor state axiom for ROBOT_CARRYING is obtained.

$$\begin{aligned} &Poss(a, s) \rightarrow \\ &[\text{ROBOT\_CARRYING}(package, do(a, s)) \equiv \\ &\quad a = \text{PICK\_UP}(package) \vee \\ &\quad \text{ROBOT\_CARRYING}(package, s) \wedge \\ &\quad a \neq \text{DROP\_OFF}(package)] \end{aligned} \tag{8}$$

i.e., the robot is carrying a package if he has just picked it up or if he was carrying it and did not drop it off. Now note for example that if ROBOT_CARRYING(P_1, S_0), then it also follows (assuming that $\text{P}_1 \neq \text{P}_2$) that
ROBOT_CARRYING$(\text{P}_1, do(\text{DROP_OFF}(\text{P}_2, S_0))$.

This discussion has assumed that there are no ramifications, i.e., sentences contributing indirect effects of actions. An example of such a sentence is $\forall s(\text{ON}(x, y, s) \rightarrow \neg\text{ON}(y, x, s))$. The assumption that there are no state constraints in the axiomatization of the domain will be made throughout this paper. In (Lin & Reiter 1994c), the approach discussed in this section is extended to work with state constraints by compiling the effects of the state constraints into the successor state axioms.

Perceptual Actions and Knowledge

To model the effects of perceptual actions, we must come up with a suitable formalization of knowledge. The approach we take is to adapt the standard possible-world model of knowledge to the situation calculus, as first done by Moore (1980). Informally, we think of there being a binary accessibility relation over situations, where a situation s' is understood as being accessible from a situation s if as far as the agent knows in situation s, he might be in situation s'. So something is known in s if it is true in every s' accessible from s, and conversely something is not known if it is false in some accessible situation.

To treat knowledge as a fluent, we introduce a binary relation $K(s', s)$, read as "s' is accessible from s" and treat it the same way we would any other fluent. In other words, from the point of view of the situation calculus, the last argument to K is the official situation argument (expressing what is known in situation s), and the first argument is just an auxiliary like the p in ROBOT_CARRYING(p, s).[2]

[2]Note that using this convention means that the arguments to K are reversed from their normal modal logic use.

We can now introduce the notation $\mathbf{Knows}(P, s)$ (read as P is known in situation s) as an abbreviation for a formula that uses K. For example

$$\mathbf{Knows}(\text{ORDERED\_SHIPMENT}(package, s) \overset{\text{def}}{=} \forall s'(K(s', s) \rightarrow \text{ORDERED\_SHIPMENT}(package, s')).$$

Note that this notation supplies the appropriate situation argument to the fluent on expansion (and other conventions are certainly possible). For the case of equality literals the convention is to supply the situation argument to each non-variable argument of the equality predicate. For example:

$$\mathbf{Knows}(\text{NUMBER}(\text{BILL}) = \text{NUMBER}(Mary), s) \overset{\text{def}}{=} \forall s'(K(s', s) \rightarrow \text{NUMBER}(Bill, s') = \text{NUMBER}(\text{MARY}, s')).$$

This notation can be generalized inductively to arbitrary formulas so that, for example

$$\exists x \mathbf{Knows}(\exists y[\text{NEXTO}(x, y) \wedge \neg\text{BROKEN}(y)], s) \overset{\text{def}}{=} \exists x \forall s'(K(s', s) \rightarrow \exists y[\text{NEXTO}(x, y, s') \wedge \neg\text{BROKEN}(y, s')]).$$

Turning now to knowledge-producing actions, there are two sorts of actions to consider: actions whose effect is to make known the truth value of some formula, and actions to make known the value of some term. In the first case, we might imagine a SENSE_P action for a fluent P, such that after doing a SENSE_P, the truth value of P is known. We introduce the notation $\mathbf{Kwhether}(\mathrm{P}, s)$ as an abbreviation for a formula indicating that the truth value of a fluent P is known.

$$\mathbf{Kwhether}(\mathrm{P}, s) \overset{\text{def}}{=} \mathbf{Knows}(\mathrm{P}, s) \vee \mathbf{Knows}(\neg\mathrm{P}, s),$$

It will follow from our specification in the next section that $\mathbf{Kwhether}(\mathrm{P}, do(\text{SENSE}_P, s))$. In the second case, we might imagine an action READ_τ for a term τ, such that after doing a READ_τ, the denotation of τ is known. For this case, we introduce the notation $\mathbf{Kref}(\tau, s)$ defined as follows:

$$\mathbf{Kref}(\tau, s) \overset{\text{def}}{=} \exists x \mathbf{Knows}(\tau = x, s) \quad \textit{where } x \textit{ does not appear in } \tau.$$

It will follow from the specification developed in the next section that $\mathbf{Kref}(\tau, do(\text{READ}_\tau, s))$. For simplicity, we assume that each type of knowledge-producing action is associated with a characteristic fluent or term in this way.

Solving the Frame Problem for Knowledge-Producing Actions

The approach being developed here rests on the specification of a successor state axiom for the K relation. For all situations $do(a, s)$, the K relation will be completely determined by the K relation at s and the action a.

For non-knowledge-producing actions, such as $\text{DROP\_OFF}(p)$, the specification (based on Moore (1980; 1985)) is as follows:

$$Poss(\text{DROP\_OFF}(p), s) \rightarrow [K(s'', do(\text{DROP\_OFF}(p), s)) \equiv \exists s' (K(s', s) \wedge (s'' = do(\text{DROP\_OFF}(p), s')))] \tag{9}$$

The idea here is that as far as the agent at world s knows, he could be in any of the worlds s' such that $K(s', s)$. At $do(\text{DROP\_OFF}(p), s)$ as far as the agent knows, he can be in any of the worlds $do(\text{DROP\_OFF}(p), s')$ for any s' such that $K(s', s)$. So the only change in knowledge that occurs in moving from s to $do(\text{DROP\_OFF}(p), s)$ is the knowledge that the action DROP_OFF has been performed.

Now consider the simple case of a knowledge-producing action SENSE_P that determines whether or not the fluent P is true (following Moore (1980; 1985)). In this case, we have:

$$\text{POSS}(\text{SENSE}_P, s) \rightarrow [K(s'', do(\text{SENSE}_P, s)) \equiv \exists s' (K(s', s) \wedge (s'' = do(\text{SENSE}_P, s')) \wedge (\mathrm{P}(s) \equiv \mathrm{P}(s')))] \tag{10}$$

Again, as far as the agent at world s knows, he could be in any of the worlds s' such that $K(s', s)$. At $do(\text{SENSE}_P, s)$ as far as the agent knows, he can be in any of the worlds $do(\text{SENSE}_P, s')$ for all s' such that $K(s', s)$ and $\mathrm{P}(s) \equiv \mathrm{P}(s')$. The idea here is that in moving from s to $do(\text{SENSE}_P, s)$, the agent not only knows that the action SENSE_P has been performed (as above), but also the truth value of the predicate P. Observe that the successor state axiom for P guarantees that P is true at $do(\text{SENSE}_P, s)$ iff P is true at s, and similarly for s' and $do(\text{SENSE}_P, s')$. Therefore, P has the same truth value in all worlds s'' such that $K(s'', do(\text{SENSE}_P, s))$, and so $\mathbf{Kwhether}(\mathrm{P}, do(\text{SENSE}_P, s))$ is true.

In the case of a READ_τ action that makes the denotation of the term τ known, $\mathrm{P}(s) \equiv \mathrm{P}(s')$ is replaced by $\tau(s) = \tau(s')$. Therefore, τ has the same denotation in all worlds s'' such that $K(s'', do(\text{READ}_\tau, s))$, and so $\mathbf{Kref}(\tau, do(\text{READ}_\tau, s))$ is true.

In general, there may be many knowledge-producing actions. Associated with each knowledge-producing action α_i is a formula $\varphi_i(s, s')$. In the case of a SENSE type of action, the formula is of the form $F_i(s) \equiv F_i(s')$, where F_i is a fluent. In the case of a READ type of action, the formula is of the form $(\tau_i(s) = \tau_i(s'))$, where τ_i is a situation-dependent term. Assume that there are n knowledge-producing actions $\alpha_1, \ldots, \alpha_n$ and therefore n associated formulas $\varphi_1, \ldots, \varphi_n$. The form of the successor state axiom for K is then as follows:

Successor State Axiom for K

$$\begin{aligned}&\forall s, s''\,(K(s'', do(a,s)) \equiv\\ &\quad[\exists s'\,(K(s', s) \wedge (s'' = do(a, s')) \wedge\\ &\quad((a = \alpha_1) \rightarrow \varphi_1) \wedge\\ &\quad\vdots\\ &\quad((a = \alpha_n) \rightarrow \varphi_n))])\end{aligned}$$

The relation K at a particular situation $do(a, s)$ is completely determined by the relation at s and the action a.

Complex Actions and GOLOG

Actions in the situation calculus are primitive and determinate. They are like primitive computer instructions (e.g. assignment). We need complex actions for the same reason we need programs.

Complex actions could be treated as first class entities, but since the tests that appear in forms like **if** ϕ **then** δ_1 **else** δ_2 involve formulas, this means that we must reify fluents and formulas. Moreover, it is necessary to axiomatize the correspondence between these reified formulas and the actual situation calculus formulas. This results in a much more complex theory.

Instead we treat complex action expressions as abbreviations for expressions in the situation calculus logical language. They may thus be thought of as macros that expand into the genuine logical expressions. This is done by defining a predicate Do as in $Do(\delta, s, s')$ where δ is a complex action expression. $Do(\delta, s, s')$ is intended to mean that the agent's doing action δ in state s leads to a (not necessarily unique) state s'. The inductive definition of Do includes the following cases:

- $Do(a, s, s') \stackrel{\text{def}}{=} Poss(a, s) \wedge s' = do(a, s)$ — simple actions
- $Do([\delta_1; \delta_2], s, s') \stackrel{\text{def}}{=} \exists s''(Do(\delta_1, s, s'') \wedge Do(\delta_2, s'', s'))$ — sequences
- $Do([\delta_1 | \delta_2], s, s') \stackrel{\text{def}}{=} Do(\delta_1, s, s') \vee Do(\delta_2, s, s')$ — nondeterministic choice of actions
- $Do(\Pi x(\delta), s, s') \stackrel{\text{def}}{=} \exists x\, Do(\delta, s, s')$ — nondeterministic choice of parameters

Other cases handle tests (ϕ?), conditionals (**if** ϕ **then** δ_1 **else** δ_2), loops (**while** ϕ **do** δ and **for** $x : \phi(x)$ **do** δ), and recursive procedures.

This set of complex action expressions forms a programming language that we call GOLOG (alGOl in LOGic), which is suitable for high-level programming of robots and software agents, as well as discrete event simulation. GOLOG differs from ordinary programming languages in that:

- it has a situation calculus semantics;
- its complex actions decompose into primitives that in most cases refer to actions in the external world;
- executing a complex action may involve arbitrary first-order reasoning (e.g., executing **while** $\exists p\,$ ROBOT_CARRYING(p) **do** DROP_OFF(p), requires inferring whether $\exists p\,$ ROBOT_CARRYING(p) in the current state) — the interpreter for this programming language is a theorem prover.

GOLOG is designed as a compromise between classical planning and detailed programming. It is a high-level nondeterministic language in which one can express schematic plans. These schemas give advice to a robot about how to achieve certain effects; without necessarily specifying in detail how to perform this action. The details are to be figured out by the theorem prover when the program is executed.

An Example

To use GOLOG to program the robot to accomplish the mail delivery task, we first need to come up with a specification of the application domain at some appropriate level of abstraction. Suppose we take the following as primitives actions:

GO_TO(*position*),
PICK_UP(*package*),
DROP_OFF(*package*), and
SENSE_REQUESTS.

The GOLOG system will essentially treat these as external procedures. Obviously, one should choose actions that can be implemented on the robot's architecture. We model states of the robot and environment using the following fluents:

ORDERED_SHIPMENT(*package*, *s*),
ROBOT_CARRYING(*package*, *s*), and
POS(*object*, *s*).

We specify the preconditions of the PICK_UP action with the following axiom:

$$\begin{aligned}&Poss(\text{PICK\_UP}(package), s) \equiv\\ &\quad \text{ORDERED\_SHIPMENT}(package, s) \wedge\\ &\quad \text{POS}(\text{ROBOT}, s) =\\ &\qquad \text{POS}(\text{OUT\_BOX}(\text{SHIPPER}(package)), s)\end{aligned} \tag{11}$$

Thus, PICK_UP is possible when someone has ordered the package to be shipped and the robot is positioned at the shipper's out-box. The precondition axiom for DROP_OFF was given earlier (2). The actions GO_TO and SENSE_REQUESTS are assumed to be always possible.

The successor state axiom for the functional fluent POS goes as follows:

$$\begin{aligned}&Poss(a, s) \rightarrow [\text{POS}(x, do(a, s)) = \vec{pos} \equiv\\ &\quad (x = \text{ROBOT} \vee \text{ROBOT\_CARRYING}(x, s))\\ &\quad \wedge a = \text{GO\_TO}(\vec{pos})\\ &\quad \vee \text{POS}(x, s) = \vec{pos} \wedge\\ &\quad ((x \neq \text{ROBOT} \wedge \neg\text{ROBOT\_CARRYING}(x, s))\\ &\quad \vee \forall \vec{pos}'\, a \neq \text{GO\_TO}(\vec{pos}'))]\end{aligned} \tag{12}$$

i.e., objects' positions are unaffected by all actions other than $\text{GO\_TO}(\vec{pos})$, which results in the robot's being at $\vec{pos}$ with everything it is carrying. The successor state axiom for ROBOT_CARRYING was given earlier (8). For the knowledge fluent K, we have:

$$\begin{array}{l} Poss(a,s) \rightarrow [K(s'', do(a,s)) \equiv \\ \quad \exists s' \, (K(s',s) \wedge s'' = do(a,s') \wedge \\ \quad\quad (a = \text{SENSE\_REQUESTS} \rightarrow \\ \quad \forall p(\text{ORDERED\_SHIPMENT}(p,s) \equiv \\ \quad\quad \text{ORDERED\_SHIPMENT}(p,s'))))] \end{array} \tag{13}$$

i.e., SENSE_REQUESTS results in the robot knowing what shipments have been ordered and the other actions have no effects on the robot's knowledge other than its knowing that they have been performed.

Which shipments are on order in a situation depends not only on the robot's actions but on what shipment requests have been made (by other agents). The robot will find out about shipment orders by sensing rather than reasoning, so we do not need to provide a successor state axiom for the fluent ORDERED_SHIPMENT. Nevertheless, we include the following axiom:

$$\begin{array}{l} Poss(a,s) \rightarrow \\ [\text{ORDERED\_SHIPMENT}(package, s) \wedge \\ \quad a \neq \text{PICK\_UP}(package) \rightarrow \\ \quad \text{ORDERED\_SHIPMENT}(package, do(a,s))] \end{array} \tag{14}$$

This allows the robot to avoid resensing whether a shipment that has not been picked up is still on order. Our current implementation handles such specifications. A completely general treatment of domains with multiple agents and exogenous events is currently under development.

Finally, we need a specification of the domain's initial state, for example:

$$\begin{array}{c} \text{POS}(\text{ROBOT}, S_0) = (5, 4, 90°) \\ \text{ORDERED\_SHIPMENT}(\text{PACKAGE}_1, S_0) \\ \text{SHIPPER}(\text{PACKAGE}_1) = \text{YVES} \\ \text{RECIPIENT}(\text{PACKAGE}_1) = \text{DANIEL} \\ \text{POS}(\text{OUT\_BOX}(\text{YVES}), S_0) = (2,3,0) \\ \text{POS}(\text{IN\_BOX}(\text{DANIEL}), S_0) = (9,5,0) \\ \text{etc.} \end{array} \tag{15}$$

Now that we have an axiomatization of the domain, we can write GOLOG programs to control the robot in terms of the primitives specified above, for example:

```
proc GRAB_ALL_LETTERS_FOR($r$)
  SENSE_REQUESTS;
  for $p$ : ORDERED_SHIPMENT($p$) $\wedge$ RECIPIENT($p$) $= r$ do
    GO_TO(POS(OUT_BOX(SHIPPER($p$)))); PICK_UP($p$)
end
proc SERVE($r$)
  GRAB_ALL_LETTERS_FOR($r$);
  if $\exists p$(ROBOT_CARRYING($p$) $\wedge$ RECIPIENT($p$) $= r$) then
    GO_TO(POS(IN_BOX($r$)));
    for $p$ : ROBOT_CARRYING($p$) $\wedge$ RECIPIENT($p$) $= r$
      do DROP_OFF($p$)
end
```

```
proc CONTROL_1
  SENSE_REQUESTS;
  while $\exists p$ ORDERED_SHIPMENT($p$) do
    $\Pi\, r$ [$\exists p$(ORDERED_SHIPMENT($p$) $\wedge$
          RECIPIENT($p$) $= r$)?; SERVE($r$)];
    SENSE_REQUESTS
end
proc CONTROL_2
  SENSE_REQUESTS;
  while $\exists p$ ORDERED_SHIPMENT($p$) do
    $\Pi\, p$ [ORDERED_SHIPMENT($p$)?;
        GO_TO(POS(OUT_BOX(SHIPPER($p$))));
        PICK_UP($p$);
        GO_TO(POS(IN_BOX(RECIPIENT($p$))));
        DROP_OFF($p$)];
    SENSE_REQUESTS
end
```

Note the use of nondeterminism in all of the procedures. CONTROL_1 and CONTROL_2 are simple examples of top-level procedures one might use to control this event-driven robot.

In the above example, we assumed that going to a given position was a primitive action. In most applications, the robot will not have complete knowledge of its environment and will occasionally run into obstacles it does not know about. Let us sketch how one could use GOLOG to write a program to get the robot around obstacles. Primitive actions that have a finer granularity than above would be used, for example:

SENSE_POS – sense the current position

SENSE_PATH(*from_pos*, *to_pos*) – run a path planner to find a path

FOLLOW_PATH(*path*) – follow *path* stopping when an obstacle is encountered

ADD_OBSTACLE($\vec{pos}$) – update the robot's map to include an obstacle

Using these primitives, a simple procedure to get the robot around obstacles can be defined as follows:

```
proc GO_TO($\vec{pos}$)
  SENSE_POS;
  SENSE_PATH(POS(ROBOT), $\vec{pos}$);
  $\Pi\, path$[PATH_FROM_TO($path$, POS(ROBOT), $\vec{pos}$)?;
        FOLLOW_PATH($path$)];
  SENSE_POS;
  if POS(ROBOT) $\neq \vec{pos}$ then
    ADD_OBSTACLE(POS(ROBOT)); GO_TO($\vec{pos}$)
end
```

This would obviously have to be extended to handle cases where no path exists or the robot gets stuck, etc. In the conclusion section, we discuss the problems of noisy sensors and inaccurate effectors.

Architectural Issues

A central objective of our project is to explore the space of possible formalisms/design methods/architectures between high-level AI approaches to

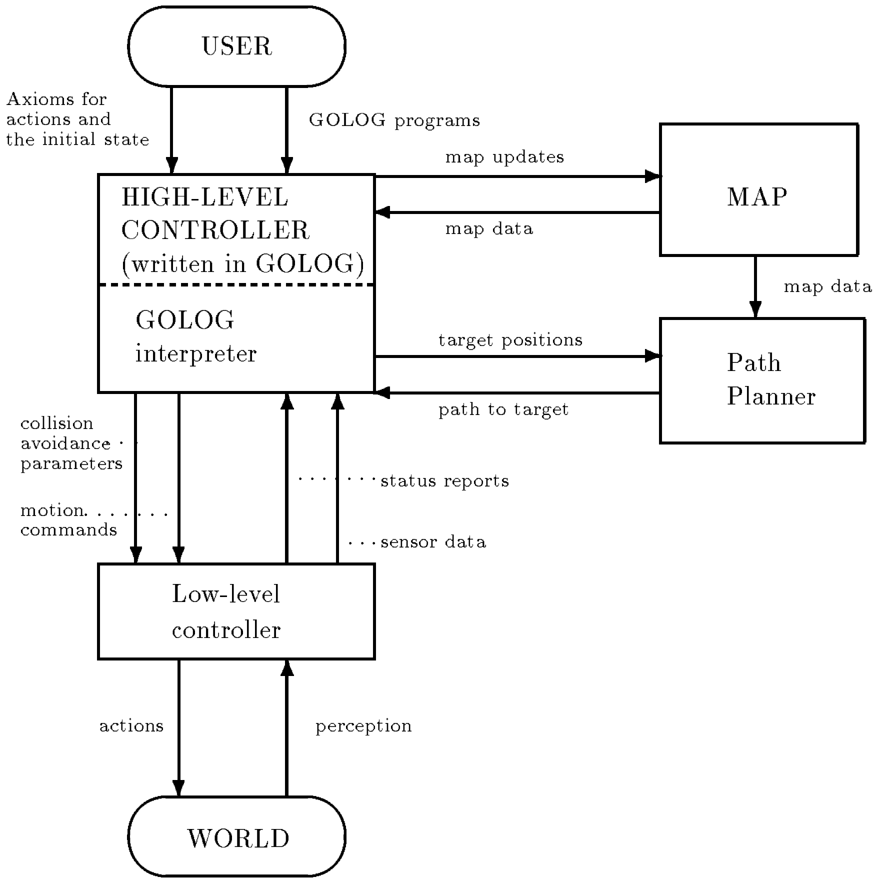

Figure 1: Proposed Architecture

reasoning about action, perception, and knowledge, and the usual robot programming methods. Let us describe one proposal for a robot architecture that arose out of our current efforts to implement a robot mail delivery application. This should not be viewed as a final answer to questions of robot architecture; different applications will require different solutions. The overall robot architecture is illustrated in figure 1. A high-level controller programmed in GOLOG drives the whole architecture. The user supplies the action precondition and effects axioms, the specification of the initial state, and the GOLOG program to control the robot; the GOLOG interpreter executes this program using the axioms to maintain the world model. The high-level controller does not drive the platform directly, but through a low-level reactive controller that performs whatever reflex actions are needed in the application, in our case, collision avoidance. This frees the high-level controller from having to respond in real-time to exceptional conditions. The high-level controller maintains a map of the environment and queries a separate path planning module to find out how to get to target positions. It then sends the paths to the low-level controller for execution. It also adjusts the parameters of the collision avoidance circuits and monitors the platform status as reported by the low-level controller. The high-level controller is responsible for getting the platform around obstacles. This might involve examining the available sensor data and perhaps performing additional sensing, determining the nature of the obstacle, updating the map (temporarily or permanently), reinvoking the path planner, and sending the new path to the low-level controller for execution. Diagnosing the type of obstacle encountered could be handled by the controller itself or by a specialized module.

GOLOG's nondeterminism presents some difficulties for an architecture that is intended to operate in the real world as opposed to simulation. In the absence of sensing acts, one can *first* completely macroexpand the GOLOG program into a sequence of basic actions that takes the initial state into a final state, and *then* execute that sequence of actions. Technically, this amounts to attempting to prove $Axioms \models \exists s Do(\mathsf{program}, S_0, s)$ and, if one is successful, using answer extraction to obtain a binding $s = do(a_n, \ldots do(a_2, do(a_1, S_0)) \ldots)$, and then executing the sequence $a_1, \ldots, a_n$. With sensing though, this is no longer generally possible because tests may depend on information previously sensed; to be able to evaluate the test, one must have actually performed the sensing. One possibility would be to expand the program until a sensing action is encountered, and at that point, execute the expanded actions; afterwards, the knowledge base would be updated with the information acquired through sensing and the whole interpretation process would start over. This appears to have two problems: first, since under this interpretation regime, sensing actions affect control flow in complex ways, it is hard to understand what a program does and hard to reason about the conditions under which it is correct; secondly, the expansion may get arbitrarily large before any action is actually performed, which leads to implementation problems. Instead, we adopt the following approach: in interpreting a non-deterministic action, GOLOG arbitrarily picks an alternative and tries to expand it until it reaches a primitive action; if this process fails because a test turns out false, then another alternative is tried; when a primitive action is successfully reached, GOLOG commits to the alternative and starts executing it. Essentially, GOLOG is expanding and executing actions one at a time, with a little bit of lookahead with respect to tests. This seems to retain the most useful aspects of non-determinism for the type of applications envisioned, especially the ability to leave it up to the interpreter to find an object that satisfies a description and perform some actions upon it. This interpretation regime can be implemented efficiently; the behavior generated by programs is easily understood and is straightforward to specify and prove correct.

GOLOG maintains a world model for the user. To evaluate the tests that the program contains, it must keep track of the effects of previous actions on world conditions, as well as revise its world model to incorporate the information acquired through sensing. In previous work (Reiter 1991; Scherl & Levesque 1993), we have proposed a method for reasoning about whether a condition holds after a sequence of actions. It uses a form of regression to reduce the given query to one involving only the initial state; the resulting query can then be handled with an ordinary atemporal theorem proving method. The approach is more efficient than plain theorem proving and is very general — it handles incompletely known initial states as well as actions with context-dependent effects. It has been proven sound and complete. Regression can also be used to incorporate the information acquired through sensing into the world model: the information acquired is regressed until it refers only to the initial state and the result is added to the theory of the initial state.

One problem with the regression approach is that its cost increases with the number of actions performed. It is clear that in general, we need to roll the agent's knowledge forward in time as actions are performed. But as Lin and Reiter (1994a; 1994b) have shown, the progression of a knowledge base (KB) need not be first-order representable. They do however identify a useful special case where the progression of a KB is first-order representable and easily computable: KB's where the state dependent sentences consist of ground literals only and where the actions have no context-dependent effects. Note that the closed world assumption is not made, so the agent's knowledge may be incomplete.

Another problem with the regression approach is that it does not handle exogenous actions; in such cases, the interpreter does not know what sequence of actions has occurred. Our current implementation handles some of these cases by doing sensing when regression cannot be performed (for instance, the fluent ORDERED_SHIPMENT in the mail delivery example). As mentioned earlier, we are working on a general account of exogenous actions.

Clearly, we should progress the KB whenever it is possible to do so and use regression as a fall back method. Work is under way to generalize the progression method. In some cases, it may be possible to partition the KB and handle some fluents by regression and some by progression. One could also have the agent strive to reduce its KB to the form required for progression by performing appropriate sensing acts. Finally, in some cases it may be necessary to be content with an efficiently computable approximation to the progression (i.e., something weaker than the strongest postcondition of the KB for the action).

Our vision for the GOLOG architecture involves the performance of a limited amount of task-level planning at run time. The user would provide a sketchy plan and GOLOG would fill out the details, the output of the planner being a complete GOLOG program. We are working on planning methods that are appropriate for agents operating with incomplete knowledge. Lespérance (1994) generalizes regression planning techniques to synthesize plans that include sensing acts and conditionals. Work is also under way to extend classical planners, both linear and nonlinear, to include control information (Lin 1994). Our experience has been that the flexibility of the situation calculus is very useful for doing this. Task planning could be triggered by an explicit call in the user's program to plan to achieve a goal. It could also arise when the interpreter notices that it does not know whether a condition tested by the program holds (in which case, the planner must find a way to achieve a knowledge goal). The planner could also be invoked upon the interpreter noticing that something has gone awry in the execution of the program, such as a precondition failing to hold; for this to work, the user would have to provide information to GOLOG as to what goal his program is intended to achieve. We have yet to decide exactly how the responsibility for the program executing successfully, including execution monitoring, exception handling, etc., should be split between the user and interpreter.

Implementation and Experimentation

A prototype GOLOG interpreter has been implemented in Prolog and various applications have been developed to test our ideas and explore architectural issues. So far all of our experiments have been run in simulation mode. The first involved a simple controller for an elevator. Various treatments of sensing were investigated. Our other experiments have involved a mail delivery application along the lines of the example presented. In one case, we interfaced the GOLOG system with the Truckworld simulator (Nguyen, Hanks, & Thomas 1994). A Truckworld domain model and GOLOG domain axiomatization were developed, and various GOLOG control programs for the robot were written.

More recently, we have been working on a more realistic version of the same application. The GOLOG system has been interfaced to software developed for the ARK robotics project (Jenkin *et al.* 1993; Robinson & Jenkin 1994). The ARK software includes a path planner, map maintenance facilities, a graphical user interface, and versions of a low-level reactive controller of the type described in the previous section for simulation, as well as for operation on platforms such as the ARK-1 (based on a Cybermotion K2A), RWI B12, and Nomad. The SENSE_PATH, FOLLOW_PATH, and ADD_OBSTACLE actions mentioned in the example correspond to operations available in the ARK software. Such operations can now be invoked directly from GOLOG. Current work focuses on developing a simple but realistic sensor configuration for the simulated robot, refining the domain axiomatization and programming an interesting range of behaviors for the application in GOLOG, as well as extending the current GOLOG interpreter with functions such as knowledge base progression and automated handling of queries by sensing and/or reasoning. We intend to progress to non-simulated experiments on some of the platforms mentioned above in the near future.

In addition to these robotics applications, we are also investigating the use of GOLOG to program software agents. These are autonomous entities that operate in large software environments such as computer networks and operating systems and assist the user by performing various tasks (Etzioni 1993). Sample tasks include monitoring bulletin boards for interesting information and compressing files that have not been accessed for some time.

Conclusion

If one looks at the history of computer programming, it is quite clear that early forms of programming were based directly on features of the underlying hardware. This is perhaps not too surprising, as the earliest programmers were either electrical engineers, or were simply forced to learn quite a bit about the (failure-prone) machinery they had to work with. In many ways, robot programming today is in a similar state. It remains very tightly coupled to robotic hardware, and to a large extent, is attempted only by individuals who are conversant with the hardware.

In this paper, we have argued for a very different form of robot programming, based on a formal theory of action and knowledge. In the long run, we expect that this approach to robot programming will enjoy much the same sort of advantages as those derived from high-level computer languages. Ultimately, we feel that it should be possible to fully control a robot with minimal attention to the details of sensors and effectors. This is certainly not the case today.

On the other hand, apart from the unreliability and other deficiencies in the hardware platforms themselves, there are a number of difficult conceptual issues that need to be resolved in our framework before it becomes practical. Among them, we have:

sensor noise: The current theory of knowledge allows us to model a sensor which acquires some value, for example the distance to the wall, to within a certain tolerance, say $\pm 20\%$. However, we cannot model the fact that it might be much more likely that the value is in the range 10–11 meters than in the range 11–12 meters, say. To model this, we need to extend the model of knowledge to incorporate (subjective) probabilities. It does appear, however, that the sort of techniques considered in (Bacchus *et al.* 1993) are readily applicable. The same applies to effector inaccuracy.

multiple agents, concurrency, and exogenous actions: Our logical framework was first developed for single agent domains. We still need to devise a true account of exogenous actions performed by others. One of the many issues this raises is concurrency: how do we solve the frame problem for concurrent actions, when the effect of two actions done in parallel need not be predictable in general from their individual effects? Another issue raised is modeling the mental life of the other agents: how does the state of knowledge of one agent, for example, change as the result of an action performed by another?

temporal reasoning and natural events: Related to the issue of exogenous action is that of natural events. We would like to be able to model situations where change of some sort is taking place unrelated to the direct action of agents, for example, a situation where water is flowing into a bathtub. This requires allowing for situations that are more like time intervals than time points, and for actions that start and end the interval (Pinto 1994).

exception handling: The form of programming we have been considering so far is most naturally viewed as directed towards some goal or goals. But during program execution, various exceptional situations might arise which also require handling. For example, we may want to have a robot slow down whenever a collision is imminent, or ring a warning bell when approaching a blind corner. We would like to develop ways of integrating this type of bottom-up processing with normal top-down execution within GOLOG.

robot-centered representations: The true primitive actions available on robots are typically robot-centered: more like "advance n centimeters" than "go to absolute location l," or like "grasp the object ahead" rather than "pick up object x." While both sorts of actions are representable in the situation calculus, the details of the relations between them need to be worked out (see (Lespérance & Levesque 1994)).

ability: Once we consider complex actions which include non-determinism and conditions whose truth values need to be determined perceptually, we also need to be concerned with whether the agent will have enough knowledge to execute the program, whether he will be able to make whatever choices are necessary (Levesque *et al.* 1994).

The topic of having the robot perform some run-time planning was also mentioned in the section on architectural issues. As far as we can tell, none of these pose insurmountable problems, and all of them admit to incremental solutions. In fact, we already have partial solutions for many of them. How all of these will be integrated in a full working robotic system, however, still remains to be seen.

Acknowledgements

The initial version of the mail delivery application and the interface between GOLOG and Truckworld were implemented by Bill Millar. We have had many useful discussions with participants in the ARK project, especially Michael Jenkin and Piotr Jasiobedzki.

References

Bacchus, F.; Grove, A. J.; Halpern, J. Y.; and Koller, D. 1993. Statistical foundations for default reasoning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 563–563. Chambéry, France: Morgan Kaufmann Publishing.

Etzioni, O. 1993. Intelligence without robots: A reply to Brooks. *AI Magazine* 14(4):7–13.

Gelfond, M.; Lifschitz, V.; and Rabinov, A. 1991. What are the limitations of the situation calculus? In *Working Notes, AAAI Spring Symposium on Logical Formalization of Commonsense Reasoning*, 59–69.

Haas, A. R. 1987. The case for domain-specific frame axioms. In Brown, F., ed., *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, 343–348. Lawrence, KA: Morgan Kaufmann Publishing.

Jenkin, M.; Milios, E.; Jasiobedzki, P.; Bains, N.; and Tran, K. 1993. Global navigation for ARK. In *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2165–2171.

Kaelbling, L. P., and Rosenschein, S. J. 1990. Action and planning in embedded agents. *Robotics and Autonomous Systems* 6:35–48.

Lespérance, Y., and Levesque, H. J. 1994. Indexical knowledge and robot action — a logical account. To appear in *Artificial Intelligence*.

Lespérance, Y. 1994. An approach to the synthesis of plans with perception acts and conditionals. In Gagné, D., ed., *Working Notes of the Canadian Workshop on Distributed Artificial Intelligence*.

Levesque, H. J.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1994. Knowledge, action, and ability in the situation calculus. In preparation.

Lifschitz, V. 1987. On the semantics of strips. In Georgeff, M. P., and Lansky, A. L., eds., *Reasoning about Actions and Plans*, 1–9. Los Altos, CA: Morgan Kaufmann Publishing.

Lin, F., and Reiter, R. 1994a. How to progress a database (and why) I. logical foundations. In Doyle, J.; Sandewall, E.; and Torasso, P., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference*, 425–436. Bonn, Germany: Morgan Kaufmann Publishing.

Lin, F., and Reiter, R. 1994b. How to progress a database II: The STRIPS connection. Technical report, Department of Computer Science, University of Toronto. To appear.

Lin, F., and Reiter, R. 1994c. State constraints revisited. To appear in the *Journal of Logic and Computation,* Special Issue on Action and Processes.

Lin, F. 1994. On goal ordering in planning: Formalizing control information in the situation calculus. In preparation.

Moore, R. C. 1980. Reasoning about knowledge and action. Technical Report 191, AI Center, SRI International, Menlo Park, CA.

Moore, R. C. 1985. A formal theory of knowledge and action. In Hobbs, J. R., and Moore, R. C., eds., *Formal Theories of the Common Sense World.* Norwood, NJ: Ablex Publishing. 319–358.

Nguyen, D.; Hanks, S.; and Thomas, C. 1994. The Truckworld manual. Technical report, Department of Computer Science and Engineering, University of Washington. Forthcoming.

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between strips and the situation calculus. In Brachman, R.; Levesque, H.; and Reiter, R., eds., *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, 324–332. Toronto, ON: Morgan Kaufmann Publishing.

Pinto, J. A. 1994. *Temporal Reasoning in the Situation Calculus.* Ph.D. Dissertation, Department of

Computer Science, University of Toronto, Toronto, ON. Available as technical report KRR-TR-94-1.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., ed., *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy.* San Diego, CA: Academic Press. 359–380.

Robinson, M., and Jenkin, M. 1994. Reactive low level control of the ARK. In *Proceedings, Vision Interface '94*, 41–47.

Scherl, R. B., and Levesque, H. J. 1993. The frame problem and knowledge-producing actions. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 689–695. Washington, DC: AAAI Press/The MIT Press.

Schubert, L. 1990. Monotonic solution to the frame problem in the situation calculus: An efficient method for worlds with fully specified actions. In Kyberg, H.; Loui, R.; and Carlson, G., eds., *Knowledge Representation and Defeasible Reasoning.* Boston, MA: Kluwer Academic Press. 23–67.